Buenas prácticas de programación en Python

Lección 3: Programación dirigida por la documentación

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Programación dirigida por la documentación

1. Presentación y objetivos

En esta lección aprenderemos a llevar a cabo una programación dirigida por una correcta documentación. Documentar el código de forma adecuada no solo facilita la tarea de terceros a la hora de comprender nuestras implementaciones, sino que también nos sirve de utilidad a nosotros mismos para depurar y analizar nuestro código.

Objetivos

- Conocer las herramientas disponibles en Python para documentar nuestro código.
- Aprender buenas prácticas para documentar nuestro código.
- Aprender con un caso práctico a documentar un programa escrito en Python.

2. DOCUMENTACIÓN: MÉTODO TRADICIONAL.

A la hora de documentar un código de programación nos referimos al acto de explicar detalladamente en qué consiste nuestro programa (a veces incluso una librería), qué funciones lo componen y cómo un usuario puede utilizar nuestro código. En términos generales podemos decir que la documentación es un conjunto de instrucciones que ayudan a un usuario a utilizar cada una de las funciones que hemos implementado en un determinado programa.

Como sabéis, Python dispone de una gran cantidad de librerías que nos facilitan nuestras tareas de desarrollo, y estas librerías pueden ser utilizadas gracias (en parte) a la documentación que sus autores han realizado. Además, la documentación de Python es considerada de buena calidad para un lenguaje de programación sin coste. El hecho de realizar una documentación de calidad es soportado por varios motivos, siendo el más importante el compromiso del creador de Python, Guido van Rossum, con el desarrollo de documentación sobre el propio lenguaje Python, así como sus librerías.

El compromiso de toda una comunidad de desarrolladores de Python permite desde la creación de informes de error a simples reclamaciones cuando la documentación realizada por otros usuarios podría ser más completa o fácil de utilizar. Este tipo de formas de comunicación y retroalimentación ha demostrado una gran eficacia a la hora de encontrar códigos documentados correctamente.

Presta atención

Es fundamental que nuestros desarrollos sean comentados y documentados correctamente, tanto para facilitar a terceros usuarios a utilizar nuestras implementaciones como para ayudarnos a comprender nuestros desarrollos en un futuro.

La documentación realizada en Python se basa en Latex y exige un conjunto de macros especialmente escritos para documentar Python correctamente. Para llevar a cabo la documentación en Python, hemos de tener en cuenta que debemos respetar una serie de directrices para crear un conjunto de directorios y ficheros asociados a la documentación a realizar.

En primer lugar, las fuentes de la documentación suelen ubicarse en un directorio "Doc" dentro de nuestro proyecto. Este directorio a su vez contiene varios ficheros y subdirectorios:

- Los ficheros han de ser autoexplicativos. Es decir, incluyen un README (léeme) y un Makefile (fichero de dependencias para la generación de la documentación final).
- Los directorios se dividen a su vez en tres categorías:

    - Fuentes de documentos. La documentación realizada en LATEX se sitúan en directorios aparte. Estos directorios tienen los nombres que se muestran a continuación:

Directorio	**Título del documento**
api/	API de Python
dist/	Distribución de módulos Python
doc/	Documentación Python
ext/	Extensión y empotrado del intérprete de Python
inst/	Instalación de módulos Python
lib/	Referencia de la biblioteca de Python
mac/	Referencia de módulos de Macintosh
ref/	Manual de referencia de Python
tut/	Guía de aprendizaje de Python

- Salida dependiente del formato. Los formatos de salida tienen un directorio asociado que contiene un Makefile que controla la generación del formato y agrupa el conjunto de documentos finales.

Directorio	**Formatos de salida**
html/	Salida HTML
info/	Salida en info GNU
paper-a4/	PDF y PostScript, A4
paper-letter/	PDF y PostScript, papel US-Letter (carta)

- Archivos adicionales. Se incluyen directorios para almacenar archivos adicionales utilizados en el proceso de generación de la documentación.

Directorio	**Contenido**
perl/	Soporte para procesado LATEX2HTML
templates/	Plantillas para los documentos fuente
texinputs/	Implementación LATEX
tools/	Guiones de procesado a medida

Importante

La tarea de documentar un código correctamente no es un ejercicio trivial. Requiere constancia y disciplina para llevar a cabo una documentación de calidad.

Sin embargo, han surgido diferentes herramientas para ayudarnos a realizar esta tarea, como Sphinx. En el siguiente apartado procedemos a explicar su funcionamiento y realizaremos un ejemplo práctico.

Como paso previo a explicar el funcionamiento de Sphinx es necesario dar unas nociones básicas sobre cómo documentar nuestro código Python. El procedimiento a seguir es muy sencillo:

1. Definimos las clases y funciones de nuestro programa.
2. Tras definirlas y antes de desarrollar su algoritmo asociado, comentamos el objetivo y funcionamiento de dicha función entre TRES comillas dobles:

```
class operacionesMatematicas:
    """Operaciones Matematicas.

    Atributos
    ----------
    op1:
        primer operando
    op2:
        segundo operando

    Metodos
    -------
    suma:
        Suma los operandos op1 y op2
    resta:
        Resta los operandos op1 y op2
    producto:
        Multiplica los operandos op1 y op2
    division:
        Divide el operando op1 entre op2

    Ejemplos
    --------
    >>> import operacionesMatematicas
    >>> ot = operacionesMatematicas(op1, op2)
    >>> resultado_suma = ot.suma()
    >>> print(resultado_suma)
    >>> resultado_resta = ot.resta()
    >>> print(resultado_resta)
    """
    def __init__(self, op1, op2):
        self.op1 = op1
        self.op2 = op2
```

```
def suma(self):
    """
    Suma los operandos op1 y op2
    """
    return self.op1+self.op2

def resta(self):
    """
    Resta los operandos op1 y op2
    """
    return self.op1-self.op2

def producto(self):
    """
    Multiplica los operandos op1 y op2
    """
    return self.op1*self.op2

def division(self):
    """
    Divide los operandos op1 y op2
    """
    return self.op1/self.op2
```

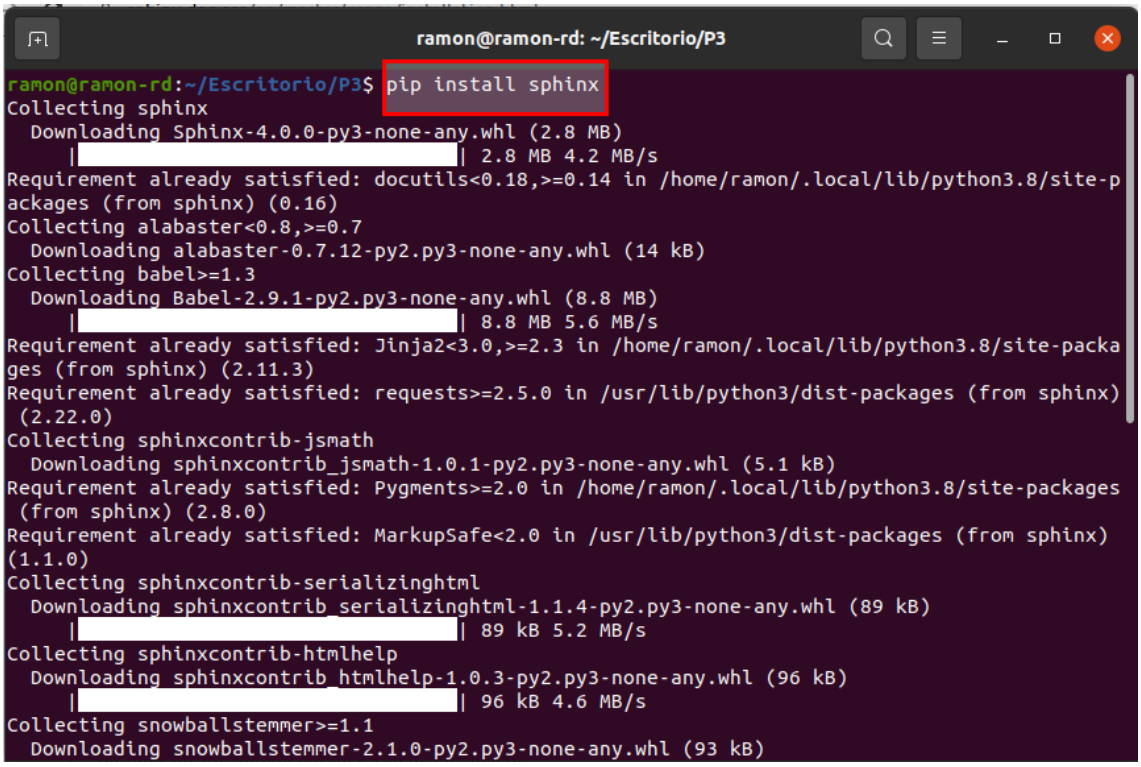
3. DOCUMENTACIÓN CON SPHINX

Sphinx es una herramienta desarrollada por Georg Barndl con el objetivo de facilitar la tarea de documentar proyectos software. Esta herramienta ha sido desarrollada especialmente para crear documentación inteligente y atractiva. Inicialmente fue creada para crear documentación de proyectos Python aunque actualmente puede ser utilizado con otros lenguajes de programación.

Hemos de tener en cuenta que Sphinx destaca frente a otras herramientas por contar con las siguientes características:

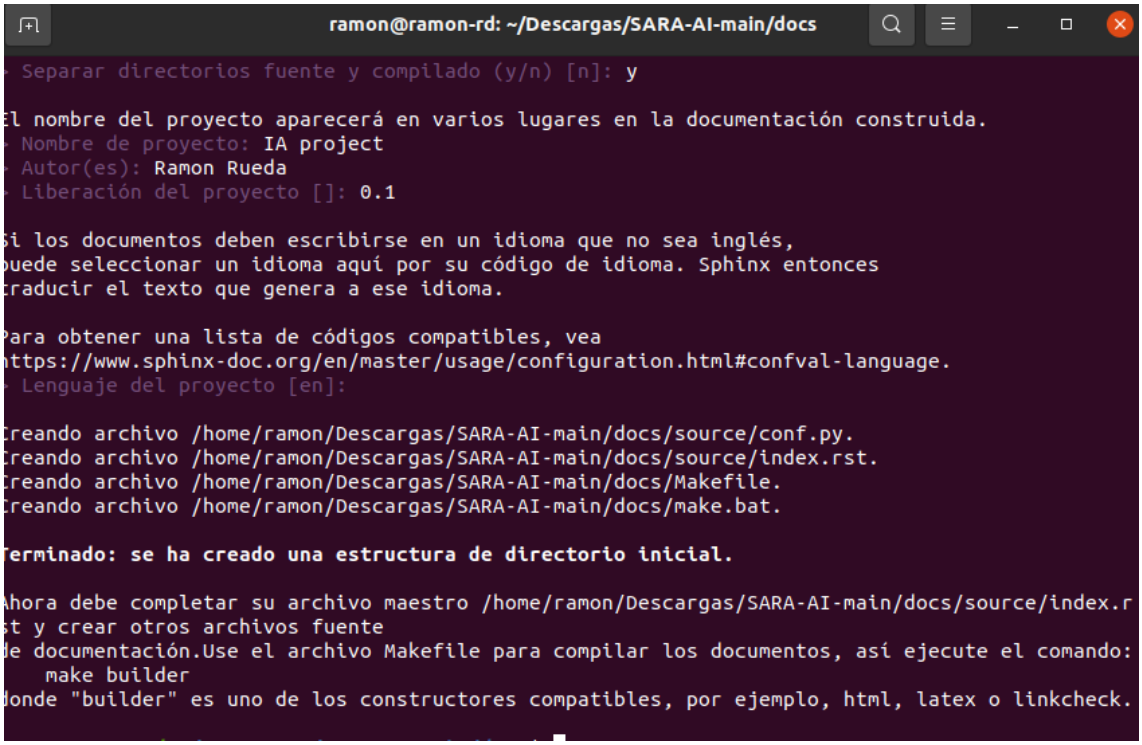
- Gran variedad de formatos de salida: HTML, LaTex, Epub, páginas de manual y texto sin formato.
- Referencias cruzadas: enlaces automáticos para funciones, clases, citas, términos de glosario, etc.
- Estructura jerárquica: permite definir de forma amigable un árbol de documentos, acompañados de enlaces automáticos a hermanos, padres e hijos.
- Índices automáticos.
- Resaltado automático de código.
- Extensiones: cuenta con diversas extensiones desarrolladas por usuarios desarrolladores, la mayoría instalables desde el repositorio oficial de python (PyPi).

Para instalar Sphinx en nuestras máquinas hemos de ejecutar el siguiente comando en una terminal (en este caso utilizamos una máquina Linux, el usuario debe adaptar el comando de instalación de acuerdo al sistema operativo que utilice):



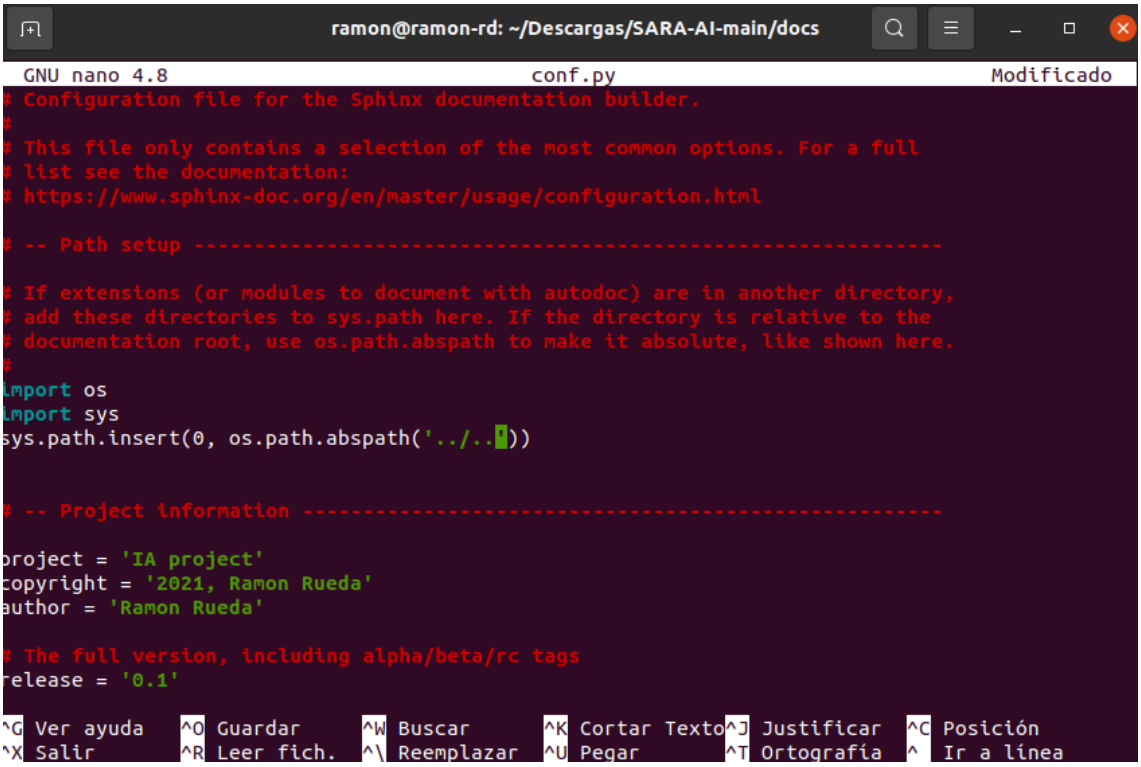
Nos dirigimos al directorio donde tenemos ubicado nuestro proyecto escrito en el lenguaje de programación Python. Una vez ahí ejecutamos sphinx para comenzar con la generación automática de documentación:

A continuación, seguimos las instrucciones que nos muestra el asistente de sphinx:

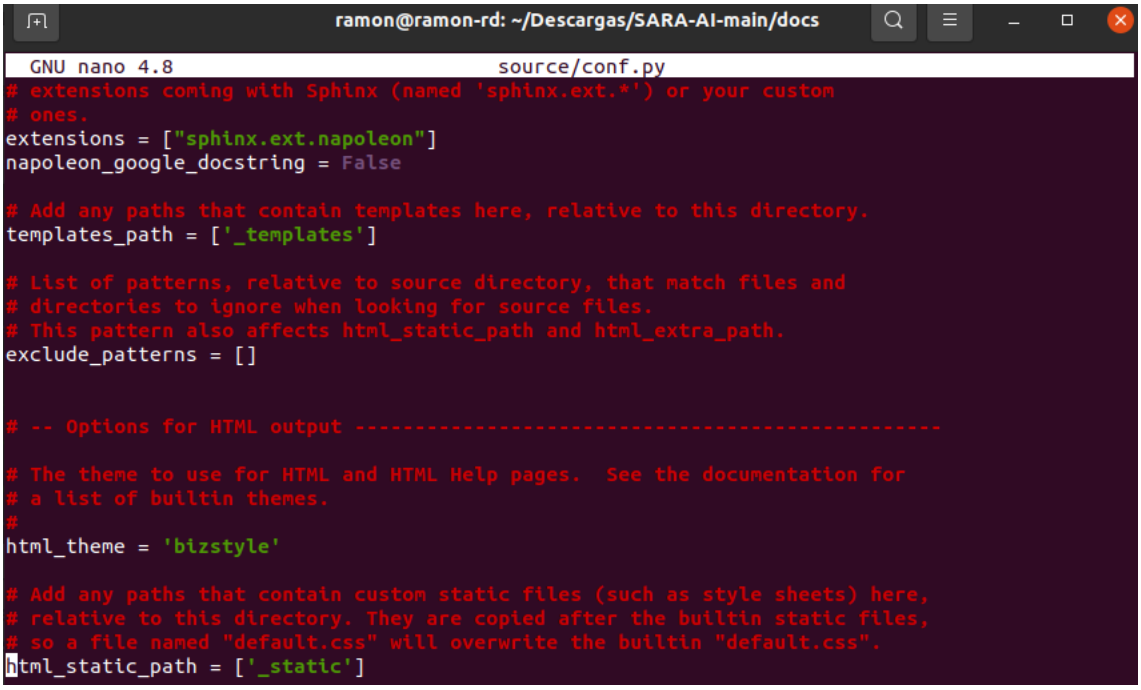


Tras finalizar el proceso, sphinx nos indica cómo compilar la documentación “make builder” aunque en este punto no habrá documentación alguna, porque todavía no se lo hemos indicado a sphinx. Para hacerlo, hemos de continuar con el siguiente proceso:

1. Establecer la ruta de nuestro proyecto. Para hacerlo, abrimos el fichero “conf.py” y descomentamos las siguientes líneas “import os”, “import sys” e “import sys.path.insert(0, os.path.abspath('.'))”. Por último, actualizamos la ruta de nuestro proyecto. En la siguiente captura se muestra un ejemplo de este proceso:



NOTA: En este fichero también podemos configurar el aspecto y diseño de nuestra documentación. En este ejemplo se han añadido los siguientes parámetros de configuración:



2. Utilizamos la herramienta sphinx-apidoc para generar los ficheros de documentación a partir de nuestro código fuente.

   sphinx-apidoc -f -o <ruta-salida> <ruta-documentar>

   Utilizamos la opción -f para sobrescribir cualquier fichero generado anteriormente.

   Utilizamos la opción -o para indicarle la ruta donde queremos almacenar los ficheros de salida.

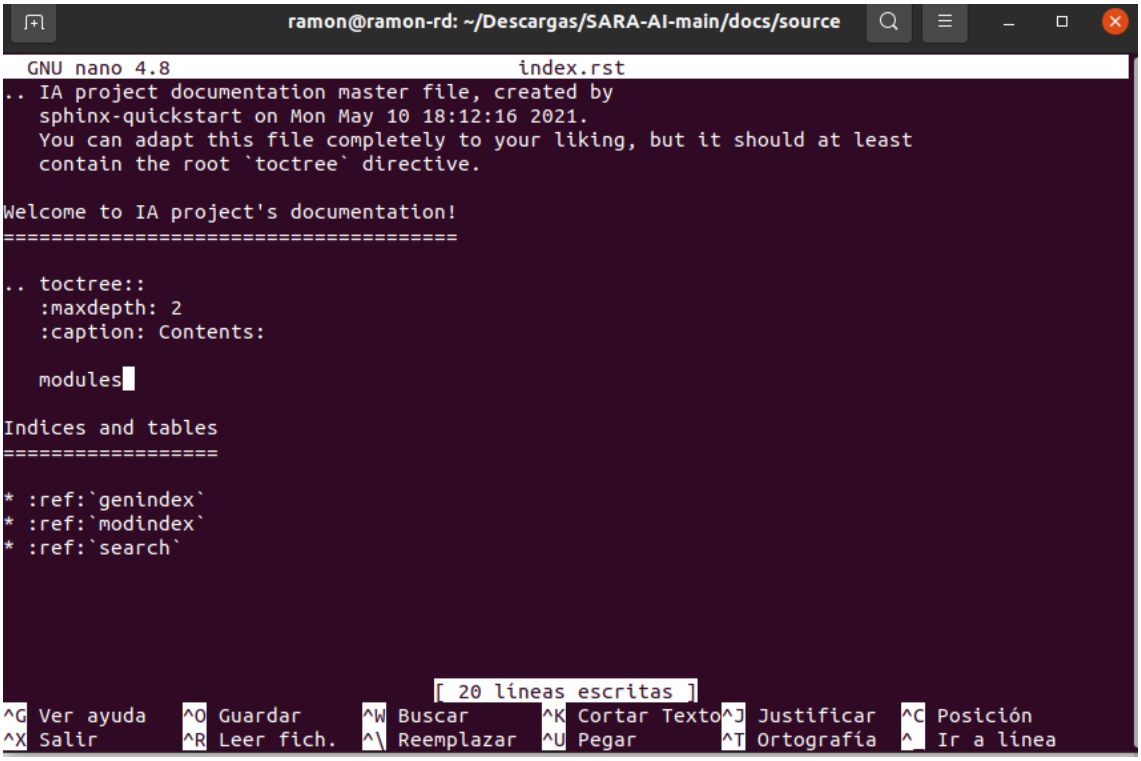
```
ramon@ramon-rd:~/Descargas/SARA-AI-main/docs$ sphinx-apidoc -f -o source/ ../trees/
Creando archivo source/trees.rst.
Creando archivo source/trees.bin.rst.
Creando archivo source/trees.binary_trees.rst.
Creando archivo source/modules.rst.
ramon@ramon-rd:~/Descargas/SARA-AI-main/docs$
```

   Dependiendo del proyecto con el que estemos trabajando se nos habrá generado un número determinado de ficheros. En este caso particular se nos han generado los siguientes ficheros:

```
ramon@ramon-rd: ~/Descargas/SARA-AI-main/docs
ramon@ramon-rd:~/Descargas/SARA-AI-main/docs$ tree
.
├── build
├── make.bat
├── Makefile
└── source
    ├── conf.py
    ├── index.rst
    ├── modules.rst
    ├── _static
    ├── _templates
    ├── trees.binary_trees.rst
    ├── trees.bin.rst
    └── trees.rst

4 directories, 8 files
ramon@ramon-rd:~/Descargas/SARA-AI-main/docs$
```

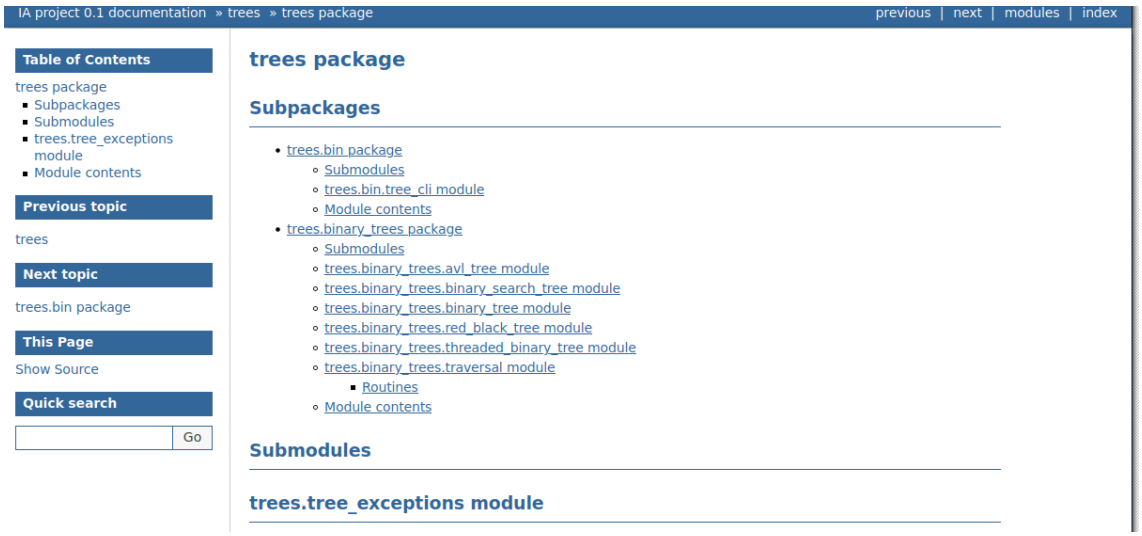
3. A continuación, editamos el fichero “index.rst” y los ficheros generados en el proceso anterior. El fichero “index.rst” es el documento “master” que sirve como página de bienvenida y contiene la raíz de la tabla de contenidos de nuestro proyecto (toctree). El “toctree” generado está vacío por defecto y hemos de completarlo con los módulos “.rst” generados.



4. Compilar los documentos. El último paso consiste en generar la documentación en el formato deseado. En este caso práctico deseamos generarlo en formato html. Para ello ejecutamos la orden “make html”.

5. Una vez finalizado el proceso, abrimos con nuestro navegador favorito el fichero index.html ubicado en la ruta:

   “nuestro_proyecto/docs/build/html/index.html”



IA project 0.1 documentation » trees » trees package previous | next | modules | index

Table of Contents

trees package
- Subpackages
- Submodules
- trees.tree_exceptions module
- Module contents

Previous topic

trees

Next topic

trees.bin package

This Page

Show Source

Quick search

Go

trees package

Subpackages

- trees.bin package
  - Submodules
  - trees.bin.tree_cli module
  - Module contents
- trees.binary_trees package
  - Submodules
  - trees.binary_trees.avl_tree module
  - trees.binary_trees.binary_search_tree module
  - trees.binary_trees.binary_tree module
  - trees.binary_trees.red_black_tree module
  - trees.binary_trees.threaded_binary_tree module
  - trees.binary_trees.traversal module
    - Routines
  - Module contents

Submodules

trees.tree_exceptions module

- trees.binary_trees.threaded_binary_tree module
- trees.binary_trees.traversal module
  - Routines
- Module contents

Submodules

trees.tree_exceptions module

Tree Exception Definitions.

exception trees.tree_exceptions.**DuplicateKeyError**(*key*)

Bases: **Exception**

Raised when a key already exists.

exception trees.tree_exceptions.**EmptyTreeError**

Bases: **Exception**

Raised when a tree is empty.

exception trees.tree_exceptions.**KeyNotFoundError**(*key*)

Bases: **Exception**

Raised when a key does not exist.

Module contents

4. PUNTOS CLAVE

En esta lección hemos aprendido:

- Cómo documentar un proyecto escrito en el lenguaje de programación Python.
- Cómo se ha llevado a cabo tradicionalmente el proceso de documentar un proyecto.
- Cómo utilizar la herramienta Sphinx para documentar nuestro proyecto escrito en Python.